# EduC – Programmer en C/Arduino - Module 6
# Le switch case et l'affichage graphique

## Introduction

Avec le `if`, on peut décider sur plusieurs conditions. Par exemple, si on lit une condition qui peut valoir seulement 4,5,8. on écrit

```
if(cond==4) {
  FaireCond1;
} else if(cond==5) {
  FaireCond5;
} else if (cond==8){
  FaireCond8;
}
```

Pour tester, allumons des Leds au hasard

```
//Edu61aIfElse.ino if-else if
// crée des nombres de 3 à 8 et on choisit la couleur
#include "EduC.h"
void setup () { SetupEduC(); }

byte cond;
void loop() {
   cond=random (3,8);  // valeurs 3,4,5,6,7
   Digit(cond);
   if (cond==4) { RougeOn; }
   else if(cond==5) { VertOn; }
   else if(cond==6) { BleuOn; }
   else { BlancOff; } // cas 3 et 7
   DelMs(300);
}
```

Le processeur passe à travers ces `if` qui ne sont pas bloquants.
S'il y a une structure `if` dans un `if` ou `else if`, cela deviens vite illisible et il y a des situations que l'on ne peut pas coder. Oublions cette façon de faire.

## Switch Case

L'instruction switch (aig) { les cas à traiter } rend l'analyse des différents cas plus claire.
aig est une variable qui numérote les cas. L'exemple précédent devient

```
switch (cond) {
   case 4:  RougeOn; break;
   case 5:  VertOn;  break;
   case 6:  BleuOn;  break;
   default: BlancOff; // autre cas
}
```

Le switch est plus efficace parce que le processeur saute dans le bon cas, et le break saute à la fin (si on ne le met pas, la cas suivant est pris, ce qui peut être très utile (par exemple si le cas 5 doit allumer le vert et le bleu).
Le programme complet est

```
//Mod62aSwitch.ino switch-case
// crèe des nombres de 3 à 8 et on choisit la couleur
#include "EduC.h"
void setup () { SetupEduC(); }

byte cond;
void loop() {
   cond=random (3,8);  // valeurs 3,4,5,6,7
   Digit(cond);
   switch (cond) {
```

```
        case 4:  RougeOn; break;
        case 5:  VertOn;  break;
        case 6:  BleuOn;  break;
        default: BlancOff; // autre cas
      }  //end switch
      DelMs(300);
    }
```

Rappelons que `switch` est traversant comme le `if`. On ne s'y arrête pas plus que quelques instructions. Le but est de surveiller tout ce qui se passe.

## Enum

Un autre grand truc est que les valeurs de la variable aiguillage peuvent être nommées. Le mot **`enum`**, est suivi par les noms donnée aux valeurs 0,1,2,... Par exemple, si on a un robot avec 2 moustaches/poussoirs,
on a 4 combinaisons 00, 01, 10, 11 qui doivent déclencher des actions et qui seront les "case" du "switch". Si on peut les nommer, c'est plus clair.

```
enum {ObsNo,ObsD,ObsG,ObsDev}; byte moustache;
switch (moustache) {
  case ObsNo:  Avance(); break;
  case ObsD:   TourneG(); break;
  case ObsG:   TourneD(); break;
  case ObsDev:  Stop(); break;
} // fin switch
```

Le programme complet doit définir les fonctions "moteur".

```
//Edu63aRobot.ino Simule un robot qui évite les obstacles
#include "EduC.h"
void setup () { SetupEduC();}

void Avance()  {LedGOn;  LedDOn;}
void TourneG() {LedGOff; LedDOn;}
void TourneD() {LedGOn;  LedDOff;}
void Stop()    {LedGOff; LedDOff;}
enum {ObsNo,ObsG,ObsD,ObsDev}; byte moustache;
//          G 1  D 2  D+G 3
void loop() {
  moustache=0;
  if (PousG) {moustache+=1;}
  if (PousD) {moustache+=2;}
  switch (moustache) {
    case ObsNo:  Avance(); break;
    case ObsD:   TourneG(); break;
    case ObsG:   TourneD(); break;
    case ObsDev:  Stop(); break;
  } // fin switch
}
```

Humm, ce n'est pas un très bon programme pour un robot. Il se bloque si l'obstacle est devant, et s'il est de coté il va y avoir des oscillations. Mais c'est facile de compléter le comportement. Par exemple:

```
case ObsDev: Recule(); DelMs(800); TourneG();DelMs(400); Avance; break;
```

## Combinaison secrète

Pour allumer la LED blanche, il faut presser les poussoirs G et D dans l'ordre D G D D G D.
Si on se trompe on programme une sirène d'alarme.
Un dessin aide à voir les états que l'on garde numérotés.
Ce que l'on numérote, ce sont les états successifs: première action, 2e action, 3e action, etc.
La première action conduit à la 2e ou à la sirène., etc. Dans chaque cas, on peut dire quel sera le cas suivant. Pour le 1er cas `if(PousD) {etat=2;} else {etat 10;} break;` // l'état 10 est la sirène
Il faut encore tenir compte des rebonds de contact des poussoirs. On va ralentir la boucle switch pour ne prendre une décision que toutes les 20ms.

```
//Edu64aSerrure.ino  Serrure
#include "EduC.h"
void setup () { SetupEduC();}

void OuvreCoffre () {} // à faire
void Alarme () {} // à faire

byte etat;
void loop() {
// on attend une pression
  Digit(etat);  // aide au test  while (PousG||PousD)   {DelMs(20);}
// L'un ou l'autre encore pressé
  while (!PousG&&!PousD) {DelMs(20);}   // Les deux relâchés
  switch (etat) {
   case 0:
      if(PousD) {etat=1;} else {etat=8;} break;
   case 1:
      if(PousG) {etat=2;} else {etat=8;} break;
   case 2:
      if(PousD) {etat=3;} else {etat=8;} break;
   case 3:
      if(PousD) {etat=4;} else {etat=8;} break;
   case 4:
      if(PousG) {etat=5;} else {etat=8;} break;
   case 5:
      if(PousD) {etat=6;} else {etat=8;} break;
   case 6:
      if(PousD) {etat=7;} else {etat=8;} break;
   case 7:
      OuvreCoffre();  while(1) ;
   case 8:
      Alarme();  while(1) ;
 }  // fin switch
}  // fin loop
```

## La minuterie d'escalier

On va pouvoir maintenant faire des minuteries d'escalier aussi complexes que l'on veut. On a vu le mode simple. Ajoutons dans le mode d'emploi :
"Economisons l'énergie! Pressez une fois pour une minute. Mais si vous avez prévu de rester plus longtemps, pressez n fois, cela durera n minutes. Inutile de prendre trop de réserve, la lampe clignotera avant de s'éteindre et vous aurez 30 secondes pour réactiver.

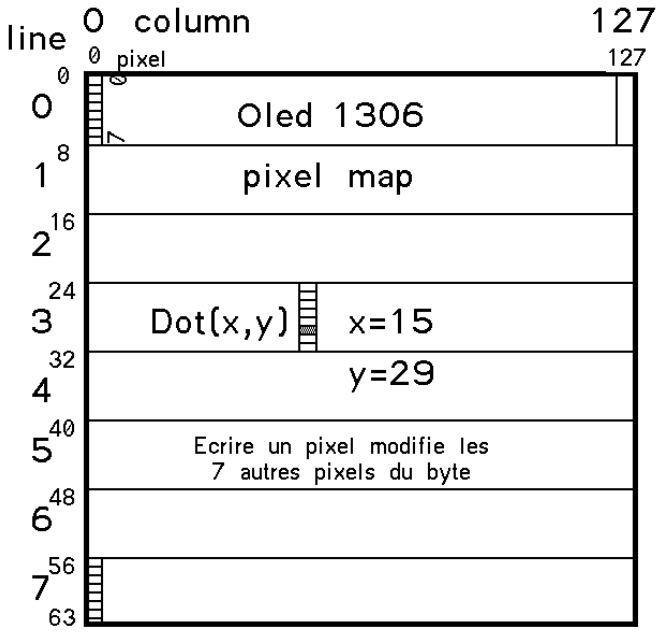
Il faut d'abord réfléchir aux variables nécessaires.
Il faut une variable `cTemps` qui décompte le temps.
Il faut un compteur de pressions `cPres` que l'on peut limiter à 10. Ce compteur augmente `cTemps` de 1 minute. Il est peut-être inutile. Un compteur de durée de pression est inutile puisque l'on compte des pressions suivies d'un relâchement.
Quels sont les états de notre "machine"?
On surveille le poussoir si c'est éteint – etat=0.
Si c'est allumé, on surveille aussi le temps etat=10.
La première pression sur le poussoir passe dans etat=1
et quand on relâche on compte et passe dans etat =2.
Dans cet etat, on decompte le temps et on surveille le poussoir. Etc..

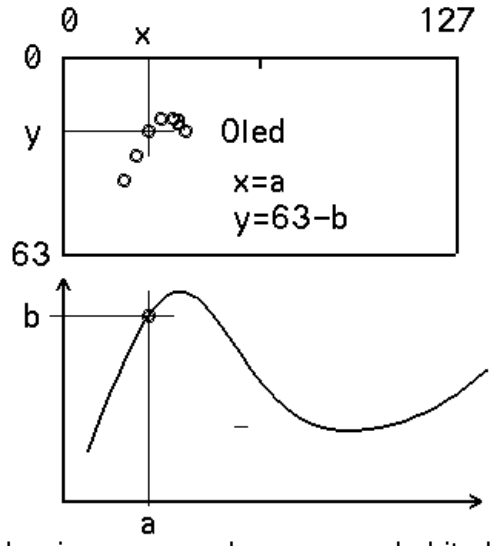
On peut faire plusieurs choses dans un état, ou quand on change d'état. Il faut bien se souvenir que l'on repasse par un état toutes les 20 ms, Si on change d'état suite à un if, une action unique peut être programmée, puisque l'on ne repassera pas par cet état au cycle suivant.

On aurait une approche similaire pour commander un robot qui doit sortir d'un labyrinthe ou ramasser des objet. Il faut faire l'inventaire des états dans lequels se trouve le robot, et dans chaque état programmer ce qui peut se passer.

## Graphique sur le Oled

L'écran est une succession de 1024 bytes qui descendent en zig-zag dans l'écran. C'est donc facile de mettre des textes et sprites de 8 de haut, un peut plus délicat en double hauteur et très délicat se ce n'est pas aligné.
La fonction Dot(x,y); place un point avec l'origine des coordonnées en haut à gauche. DDot(x,y); fait un point de double hauteur. ! byte x,y; est défini dans la librairie. Il ne faut pas le redéfinir




Pour dessiner une courbe avec nos habitudes de représentation, il faut saturer à 63 et faure un complément é 63 pour y. Pout x, on sature ou recommence au début de l'écran.

Autre fonction définies dans Oled.h: Hline (y); Vline (x); Ball(x,y); Raq(x,y,h);

## Exemple

```
//Edu65aDessin.ino    Dessin au potentiomètres
#include "EduC.h"
#include "Oled.h"
void setup () { SetupEduC(); SetupOled(); }

int aa,bb;
void loop(){
aa= GetPotG(); LiCol(0,0); Hex8(aa);
bb= GetPotD(); LiCol(0,100); Hex8(bb);
// GetPot 0--255  ecran 0--63
  y= 63-(aa/4);
  x= 63-(bb/4);
  Dot (x,y);
}
```

On ne peut pas faire un joli dessin comme cela. Il faudrait pouvoir "lever la plume" (par exemple PousG PousD) et effacer (les deux poussoirs en même temps).

## Statistique

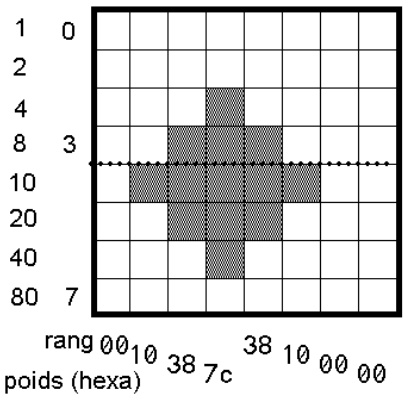
Ce programme tire un nombre au hasard (0-127) et augmente le y pour le x trouvé. Le programme s'arrête quand un maximum de 63 est atteint. Les lignes horizontales sont dues à une simplification dans la routine Dot();
Enlever le randomSeed(GetPotD); pour vérifier que le dernier point reste au même endroit, donc toute l'évolution a été la même.

```
//Edu66aStatistique.ino
#include "EduC.h"
#include "Oled.h"
void setup () { SetupEduC(); SetupOled();randomSeed(GetPotD); }

byte taHisto[128];
byte mesure,yy,moy;
int cnt;
void loop() {
  mesure = random (0,64);
  yy = ++taHisto[mesure];
  Dot (mesure*2,63-yy);
  if (yy==63) {while(1);} //fini
  delay (10);
}
```

## Lutins

Un lutin (sprite) doit être défini dans une table. Cette table est en librairie pour les sprites Smile,et Sad que l'on a rencontré, appelés par Sprite(Smile);
Si le sprite est dans la mémoire programme de l'utilisateur, comme la croix ci-contre, il faut déclarer
`byte croix[]= {0x00,0x10,0x38,0x7c,0x38,0x10, 0x00,0x00};`
L'instruction pour copier à partir du pointeur est
`MySprite[croix];`
La longueur est évidemment inférieure à 127.




Pour un sprite de 16 de haut, il faut déclarer 2 sprites de 8 de haut et les aligner dans la fonction qui appelle deux fois MySprite();
Plus de détails sous www.didel.com/Oled1306.pdf

```
void BigSmile(byte li,byte col) {
   Licol(li,col); MySprite(smiletop);
   Licol(li+1,col); MySprite(smilebot);
}

void loop() {
   BigSmile(3,50); // vers le centre
   while(1);
}
```